型付き関数型言語におけるより柔軟な外部データとの連携方式の研究

修士学位論文要約（令和 2 年 9 月）

型付き関数型言語におけるより柔軟な外部データとの連携方式の研究

高城　光平
指導教員：大堀 淳，　学位論文指導教員：上野 雄大

A Study on dealing with external data in a statically typed functional language

Kouhei TAKAGI
Supervisor: Atsushi OHORI,　Research Advisor: Katsuhiro UENO

Practical programs need data stored in the outside of the program. The data are called as “external data” in the sense that their format are different from the one used in program languages. External data are collected independently of programs and therefore they have different meanings from the interpretation that programming languages define. This means that external data cannot be imported to a program with keeping their accurate meanings. To reconcile the gap, the goal of this study is to develop a mechanism that gives external data an flexible interpretation to the values defined in programming languages. Toward this goal, in this study, we focus on numeric values and null values in JSON data and extend the SML# compiler with the following functionalities: (1) the number type, which allows the programmer to import JSON’s numeric values with its accuracy and to convert them to a value of SML#, (2) the `with null of` construct, which replaces each null value with an SML# value before importing the JSON data.

1. 序論

今日，実世界で生成される種々の情報は何らかの方法によって，プログラムとは独立に収集され，外部データとして保存される．外部データは，独自の意味や表現を持ち，個々のプログラムはこれら種々のデータを読み込むことによって，プログラムが利用可能な内部データに変換する．既存のプログラミング言語の多くは，外部データをサポートする機能を提供する．しかし，外部データ記述言語とプログラミング言語の間には，どちらかに固有の値や値の表現形式の違いなど，いくつかの解決すべき課題がある．これにより，既存のプログラミング言語が外部データを扱う際，データのもつ意味や解釈を含めた相互変換は可能ではない．

例えば，JSON データをプログラミング言語で扱うことを考える．多くの言語では，外部データをその言語の内部データとして取り込む際，一度処理プログラムに合わせた形式に変換し，変換された外部データをプログラムが読み取って利用する．しかし，この方法では，JSON の数値データの解釈を正確に反映しているとは言えない．なぜなら，JSON データは機械および人間の双方による理解が比較的容易な表現形式で記述される [1, 2] 一方，静的型付き言語はコンパイル時に値の型情報を一意に解釈する性質をもつからである．

また，外部データには固有の値である null 値が存在する．null 値は，不明な値や欠損を表す特殊な値であり，プログラミング言語は null 値を扱う機構を備えていない．現状，null 値に対する系統的なアプローチは存在しないが，ML の option 型や C#の Nullable 構造体などを用いてアドホックに扱う方法もある．しかし，null 値を含めた構造データをこのような方法を用いて内部データとして取り込み，プログラムがその値を要求することは，外部データの解釈を十分に理解しているとは言えない．

本研究は，外部データの解釈をプログラムの実行時にユーザが与える機構の構築を目指す．これを達成するために，対象とする外部データを JSON に限定して，SML#[3] を拡張し，以下の機能を提供する．：(1) 数値型の値 number を受け取り，SML#の任意の数値型に変換する機能，(2)null 値の処理を記述し，対応する箇所の null 値を変換する機能．

2. 本拡張に必要な SML#の機能

本研究は SML#コンパイラが提供する動的型付け機構を基盤とする．動的型付け機構は以下のコンポーネントからなる．

- ML が表現可能な型の値に対応する，その型情報を含むデータ構造 `reifiedTerm`
- 型付きデータと `reifiedTerm` 間の相互変換 (`toReifiedTerm` 関数)

- JSON データを表現可能になるよう拡張された `reifiedTerm`
- JSON データと `reifiedTerm` 間の相互変換（`reifiedTermToML` 関数）

この機構による JSON データの処理は，以下のように行われる．まず JSON データを受け取って構文解析し，その構造に対応するデータ構造 `reifiedTerm` を与えて動的な値を生成する．次に，動的な値の静的なデータ構造を動的に型検査することで ML の型情報を与え，ML の値に変換する．

3. ユーザが実行時に解釈を与える機構の導入

本章では，本研究で実現する言語機構の概要を説明する．

3.1. 数値データにユーザの解釈を与える機構

JSON の数値データの複数の解釈を SML#コンパイラで扱うために，以下の拡張を行う．まず，JSON の数値型 number に対応するデータ型 `JSON.NUMBER` と対応するデータ構造 `ReifiedTerm.NUMBER` を追加する．ここで，JSON の数値の解釈を可能な限り保持するために，`ReifiedTerm.NUMBER` の表す値は文字列とする．次に，`JSON.NUMBER` 型の値を構文解析すると `ReifiedTerm.NUMBER` の値が得られるような相互変換を追加する．その後，`ReifiedTerm.NUMBER` 型の値を SML#の任意の数値型の値に変換できるよう，`reifiedTermToML` 関数を拡張する．

`reifiedTermToML` 関数は，以下の手順で `ReifiedTerm.NUMBER` 型の値を ML の値に変換する．まず，`ReifiedTerm.NUMBER` 型の値を浮動小数点数 decimal_approx 型の値に変換する．ただし，この値は `sign:bool`，`digits:int list`，`exp:int` をフィールドにもつ．次に，decimal_approx 型の値をユーザが指定した ML の数値型に応じて変換する．このとき，ユーザが実数型を指定した場合，関数 `fromDecimal: decimal_approx -> real` を用いて変換する．整数型の場合，以下の処理を行うことで変換する．まず，`exp` が正値ならば，`digits` から `exp` 個分の要素を取り出し，`exp` が負値ならば，`digits` の長さ $>$ |exp| の場合，その差分の個数を `digits` から取り出して数字列を生成する．`digits` の長さ $<$ |exp| の場合，”0”を返す．次に，ここで得られる数字列を ML の intInf 型の値に変換した後で，ユーザが指定した整数型への変換を行う．ここで，`exp` $>$ `digits` の長さならば，その差分回だけ 10 を乗じた数を掛け，桁上げを行う．

3.2. null 値にユーザの解釈を与える機構

null に解釈を与えるために，動的型付け機構を以下のように拡張し，図 1 に示す構文を導入する．この構文では，`with null of` 以下に null 値に対するユーザの解釈と置換したい値を記述する．ここで，

`_dynamic` dyn `as` τ
　`with null of` ty_1 `=>` exp_1 `|` $\cdots$ `|` ty_N `=>` exp_N

図 1　null に対する解釈を実行時に与える構文

ty_i には null 値の解釈を ML の任意の型で記述し，exp_i には null に代替する値を記述する．（ただし，$1 \leq i \leq N$.）この構文を実行すると，dyn として受け取った動的な値の τ の範囲にいずれかの ty_i にマッチする null が含まれるかを後述する手順によって検査する．ここで，動的な値 dyn は，JSON データのデータ構造に対応する `reifiedTerm` 表現から生成される値であり，`dynamic` 型の値として表現される．さらに，そのような null が存在するならば，null が存在する箇所の値を exp_i の `reifiedTerm` 表現に置換する．もし，マッチする null が存在しないならば，SML#コンパイラは null 例外を返す．これらの処理を τ のすべての範囲を検査し，null が置換された後の動的な値に対して，τ 型を与えることで，ML の値に変換する．

ここで，`with null of` 以下の null の解釈にマッチする JSON データに含まれる null 値を走査し，ユーザ指定の値に変換する処理は以下のように行われる．まず，dyn から τ 型の ML の値を生成する関数に，dyn，τ および ty_i と exp_i のペアのリストを渡す．次に，dyn から `reifiedTerm` 表現を取り出し，τ の範囲に null が存在するかを検査し，存在しなければ何もしない．存在する場合，null が位置する τ の型が ty_i と一致するかを検査する．

4. 結論

本研究では，静的型付き言語が JSON データを扱う上で生じる問題点を分析し，その解決策を提案した．さらに，JSON の数値データおよび null 値に対する解釈をユーザがプログラムの実行時に与える機構を SML#に実装した．このとき，数値データに対応するデータ構造を追加し，ML の任意の数値型の値に変換する処理を追加した．null 値に対し，その変換規則を記述可能な新たな構文を導入し，ユーザ指定の範囲に含まれる null 値を走査し，値を置換する処理を追加した．今後は，ユーザ定義のデータ型との対応を議論していきたい．また，JSON 以外の外部データとの対応を考えたい．

文献

1) Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, 2006.

2) The JSON Data Interchange Syntax. `https://www.iso.org/standard/71616.html`.

3) SML# Project. `http://www.pllab.riec.tohoku.ac.jp/smlsharp/`.